A case-comparison study of automatic document classification utilizing both serial and parallel approaches

B Wilges[1], R C Bastos[1], G P Mateus[2] and M A R Dantas[2]
[1] Department of Engineering and Knowledge Management (EGC)
Federal University of Santa Catarina (UFSC)
Florianópolis, SC, 88040-900, Brazil
[2] Department of Informatics and Statistic (INE)
Federal University of Santa Catarina (UFSC)
Florianópolis, SC, 88040-900, Brazil

E-mail: beaw@inf.ufsc.br

Abstract. A well-known problem faced by any organization nowadays is the high volume of data that is available and the required process to transform this volume into differential information. In this study, a case-comparison study of automatic document classification (ADC) approach is presented, utilizing both serial and parallel paradigms. The serial approach was implemented by adopting the RapidMiner software tool, which is recognized as the world-leading open-source system for data mining. On the other hand, considering the MapReduce programming model, the Hadoop software environment has been used. The main goal of this case-comparison study is to exploit differences between these two paradigms, especially when large volumes of data such as Web text documents are utilized to build a category database. In the literature, many studies point out that distributed processing in unstructured documents have been yielding efficient results in utilizing Hadoop. Results from our research indicate a threshold to such efficiency.

1. Introduction

The intensive utilization of the web has created an unbelievable volume of information never seen before. Inside the World Wide Web environment there are three classes of data: unstructured, semi-structured and structured data. The first type of data is represented by the HyperText Markup Language (HTML), semi-structured by the Extensible Markup Language (XML), and, finally, conventional databases can represent structured data. As mentioned in the report from the Royal Pingdom [1], an Internet monitoring company, in December 2011 the number of sites was circa 555 million, with an increase of 300 million added only in 2011. In this year, other interesting examples are the growth of Web servers utilizing Apache technology, which was equal to 239.1% and the number of Internet users reached 2.1 billion. These numbers highlight the required efforts for retrieving and classifying documents from a specific subject. A study by Autonomy [2], a Hewlett-Packard Company, mentions that the amount of unstructured data corresponds to 90% of the entire information set.

It is interesting to notice that the actual computational scenario, where studies on mobile configurations and lightweight protocols are growing [3][4], it is expected that the amount of data

could increase in an unprecedented trend. This argument shows the importance of new retrieval techniques [5] and other experiences as key elements to provide some assessment to organizations.

As observed by Yan and Pederson [6], the efficiency of data retrieval is related on how the document database is configured. Thus, if a document exists within a large volume of data, inside a pre-classified set of data, it will probably be more efficient to gather specific information. In contrast, a natural assessment of large amounts of data would be difficult for humans due to the large amount of information available. Therefore, computational environments are more appropriate to process massive information sets. As Russel and Norvig [7] point out, these types of natural language processing are usually being executed through automatic document classification (ADC).

As mentioned in Baeza and Yates [8], the area of Information Retrieval (IR) has been the goal of many studies seeking automatic techniques capable to search and organize text documents using natural languages, including unstructured data. These approaches aim to gather, from user query, differentiated information available in libraries, storage systems, and specific places in order to yield dissemination of information.

This paper is characterized by an empirical case study, where the main objective is to present one assessment related to performance during the text pre-processing phase to build term bases from diffuse automatic document classification (ADC). Therefore, in this research we show a case-comparison between serial and parallel approaches. The serial element is implemented by adopting the RapidMiner software tool [9], which is recognized as the leading open-source system for data mining. On the other hand, the parallel side is represented by the MapReduce programming model (2013), utilized in the Hadoop [10] software environment.

Therefore, this paper is thus organized: theoretical aspects of RapidMiner, MapReduce, Hadoop, and HDFS are presented in Section 2. Section 3 presents the proposal and environment related to the present investigation. Empirical results based on experiments are presented and discussed in Section 4. Finally, in Section 5, conclusion and suggestions for further studies are presented.

2. Theoretical Aspects

In this section, we shall explore some conceptual aspects related to RapidMiner, MapReduce, Hadoop, and HDFS technologies. These four techniques are key elements for the proposal of this research work.

2.1. RapidMiner

RapidMiner [9] is an open-source system for data mining. It is available as a stand-alone application for data analysis and as a data mining engine for integration into other products. Furthermore, thousands of installations of RapidMiner in more than 40 countries give these users with competitive advantage [9].

Some essential characteristics of the RapidMiner [9] are:

- Data integration, Analytical ETL, Data Analysis, and Reporting in one single suite;
- Powerful but intuitive graphic user interface for the design of analysis processes;
- Repositories for process, data, and meta data handling;
- Single solution with meta data transformation: no trial and error, inspecting results directly during design stage;
- Single solution with on-the-fly error recognition support and quick fixes;
- Complete and flexible, with hundreds of data loading, data transformation, data modeling, and data visualization methods.

RapidMiner [9] is also well-known for some other operational features such as:

- Freely available open-source data mining and analysis system;
- Executes on every major platform and operating system;
- Most intuitive process design;
- Multi-layered data view concept which ensures efficient data handling;

- GUI mode, server mode (command line), or access via Java API;
- Simple extension mechanism;
- Powerful high-dimensional plotting facilities;
- Solution available for more than 500 operators for data integration and transformation, data mining, evaluation, and visualization;
- Automatic meta optimization schemes;
- Definition of re-usable building blocks;
- Standardized XML interchange format for processes;
- Graphic process design for standard tasks, scripting language for arbitrary operations;
- Machine learning library Weka [11] fully integrated;
- Access to data sources such as Excel, Access, Oracle, IBM DB2, Microsoft SQL, Sybase, Ingres, MySQL, Postgres, SPSS, dBase and Text files;
- Comprehensive data mining solution with respect to data integration, transformation, and modeling methods.

According to the RapidMiner [9] manual, it succeeds in nearly all cases in automatically determining the correct execution order of operators. In order to do so, RapidMiner uses information connection and an operator, the result of which is to be used by another operator, must obviously be executed before it. This works particularly well for this study as it is a sequential approach, since when the process was created its operators were properly lined up one after the other. There are other versions of RapidMiner software that implements parallel processes; however, in this research the free RapidMiner version has been used, with access to serial approaches only.

2.2. MapReduce

The actual world scenario causes the majority of organizations to deal with a requirement to process an enormous variety of data, of different types. Such processing is characterized by an information extraction operation from these data and aggregation of knowledge to the organization. The importance of such processing can be observed through two major aspects: the first, more evident, is the high volume of data; the second is related to the type of data, that is unstructured.

MapReduce is a programming model created by Google, typically used for processing large data sets with distributed configurations. The basic idea is to map and then reduce large tables, similarly to usual functional programming. However, it is different from a function programming language, since in the end MapReduce does not have the same original forms. The strength of the MapReduce programing model leads to the existence of several libraries that have been written in many programming languages.

As Lammal [12] states, the programming model of Google MapReduce is designed to process a large volume of data in a parallel fashion. Data processed by MapReduce are typically unstructured, such as texts and images. The parallel characteristic of this model allows a solution for any problem, dividing these into data matrixes which are executed without necessarily communicating among parallel tasks. In addition, as pointed out by Lammal [12], the programming model is based on simple concepts: input interaction; calculus for the peer key/value for each input; gathering of all intermediate values through a key use; interaction among resultant groups; reduction of each group.

As Google Code University [13] states, MapReduce was developed inside Google as a mechanism to process large amounts of data, in which only a distributed high performance system could manage to process such volume of data within reasonable elapsed time. In other words, this means utilizing a parallel paradigm. The approach provides an abstraction that allows for simple calculus of several problems, hiding details related to parallelization, data distribution, load balancing, and fault tolerance.

An open-source example of MapReduce is the implementation of Apache called Hadoop [10]. Hadoop utilizes a file system named Hadoop Distributed File System (HDFS) to store unstructured data. The Hadoop implementation allows applications execution in high-performance distributed configurations, with thousands of nodes and fault tolerance facilities.

2.3. Hadoop

The Hadoop is a project from the Apache [10] characterized as an open-source framework implemented in Java language, which provides facilities for the development of distributed applications. Characteristics of these distributed applications are the requirement of large amount of data (reaching sometimes petabytes) and thousands of computer nodes.

The Hadoop project is based upon the Google File System (GFS) [14] and MapReduce. In 2006, the developer of Hadoop joined the Yahoo team and from this date the project is one entire system that works from the Web. In 2008, the project became an independent effort inside Apache and has been developed by a large community of programmers. The success of Hadoop can be measured by the number of large companies that are adopting the solution; some examples are: Facebook [15], Last.fm [16], Twitter [17], IBM [18], and Microsoft [19]. In addition, as an open-source effort in the official page of the project [10], it is possible to find JAR files necessary for Hadoop startup. In the Hadoop website, source codes, documentation, and several contributions can also be found.

2.4. Hadoop Distributed File System (HDFS)

Nowadays, distributed file systems are one of the most important research topics inside the area of distributed systems. Examples of these efforts are Google File System [14], Amazon Simple Storage Service [20], Ceph [21], Fraunhofer File System [22], Lustre [23], IBM General Parallel File System [24], and the Parallel Virtual File System [25].

The Hadoop Distributed File System [26] is a scalable and portable distributed file system written in Java for the Hadoop framework. HDFS is based on the GFS and has master/slave style architecture. The TCP/IP [4] stack protocol supports HDFS, in which a client uses the ClientProtocol with the NameNode through a port. DataNodes adopt the DataNodeProtocol with the NameNode, and these protocols execute a Remote Procedure Call (RPC).

An HDFS cluster comprises a unique NameNode, a master-server which is responsible for the system file management and controls access to all client files. On the other hand, there is a set of DataNodes, usually one per cluster node, which acts as manager for local storage. The NameNode operates all commands in the file systems, such as open, close, and rename (file or directory). Figure 1 below presents a typical HDFS architecture.

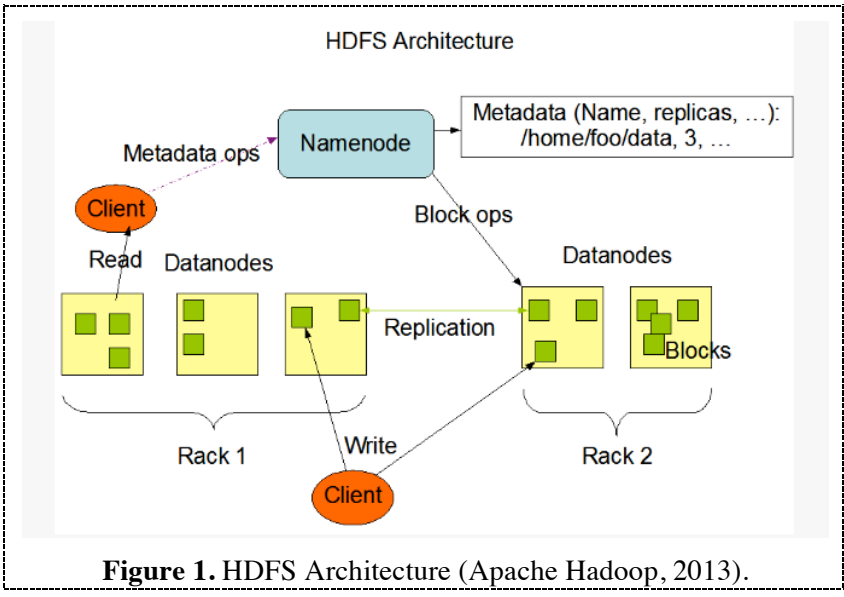

Figure 1. HDFS Architecture (Apache Hadoop, 2013).

HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. A typical block size

used by HDFS is 64 MB large. Reliability of the file system is reached through replication of data in several other nodes. Thus, it avoids the requirement of RAID storage.

HDFS does not provide high availability, because HDFS configuration requires one server (NameNode), thus this node represents a unique point that can fail. In case of failure, when server startup occurs it will be necessary to process all pending operations. This restart procedure can reach up to 30 minutes for large cluster environments. This period could be shorter in case a secondary NameNode has a procedure to gather regularly information in small time intervals about the NameNode master. As a result, the NameNode server is not obligated to execute all pending actions.

3. Proposal and test bed environment

The massive increase in data volume created with the intensive utilization of Web applications creates a new computational scenario where any conventional analysis of large-scale datasets has become outmoded.

In this section, we present the idea behind our study, which is to acquire knowledge on data processing by using an open-source system and then to consider a speedup technology for this process. Later, we have applies this assessment for an empirical process of text pre-processing to build term bases from a diffuse automatic document classification (ADC).

In order to reach our goal, we show a case-comparison study between two software environments, with different paradigms: one serial and one parallel. The serial software package is represented by the RapidMiner software tool [9], and the parallel paradigm is represented by the MapReduce programming model [27] utilized in the Hadoop [10] software environment.

Goller, G., J. Loning, T. Will, W. Wolff [28] can distinguish two stages in automatic document classification: the learning stage and the subsequent classification stage. In the learning stage, users define categories in which they are interested by giving sample documents (training examples) for each of these categories. Furthermore, the process of document classification involves three different steps: text preparation, characteristics selection, and later category definition.

Text preparation in the ADC approach is a relevant and costly step. This arguments is based on the required transformation process of the document set, considering a natural language, in a useful list of terms and also take into account a compatible form to prepare to extract knowledge.

In this research, as a test bed environment, we have first utilized resources from RapidMiner [9]. Figure 2 below shows a layout related to the text processing procedure, illustrating the interface feature from the environment, (claimed by RapidMiner developers), where they state that the software tool as a powerful and intuitive graphic user interface for the design of analysis process.

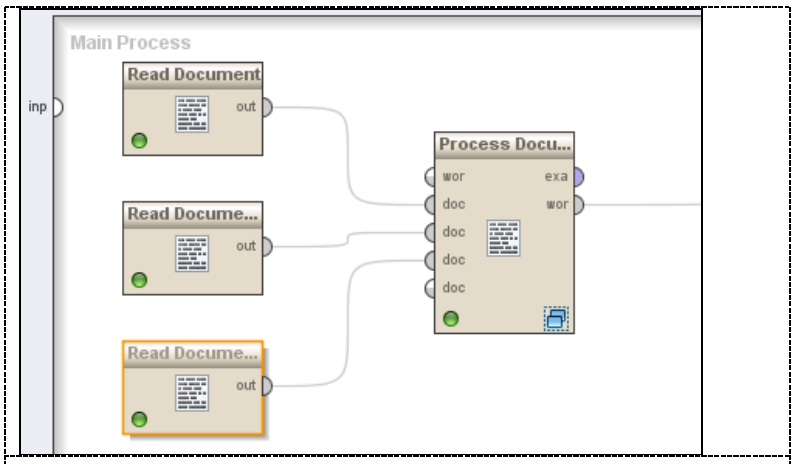

Figure 2. Example of text reading and processing.

On the other hand, Figure 3 below presents details of the text processing procedure illustrated in Figure 2 above. All texts from the same category are required to go through this pre-processing step in order to build a database which will provide storage facility for the most frequent terms and its respective grade of relevance for the category.

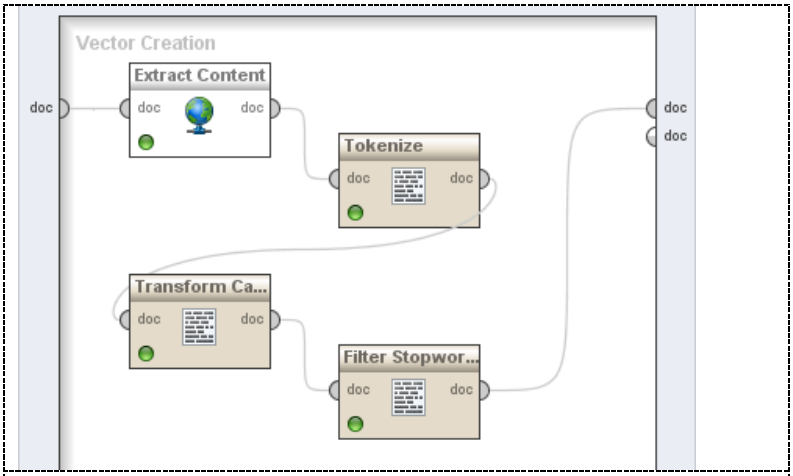

Figure 3. Detailing of document processing.

Figure 3 indicates that document processing is characterized first by the Extract Content operator, which extracts textual contents from a HTML document and yields text blocks extracted from the document. In sequence, the Tokenize operator divides text from a document in a token sequence. The next operation, Transform Case, transforms all characters from a document to uppercase or lowercase. Finally, the Filter Stopwords (Dictionary) operator extracts all words that are inside a stop-words list. This list is uploaded from a specific file inside the same operator. The stop-words list contains all words that are not relevant for the count of frequent terms inside the text document. In other words, this means that those words do not provide any influence for the definition of a text category.

In the second moment of the present study, targeted to build a database of a category with differentiated performance, we have considered the use of the MapReduce technique from the Hadoop software environment. In other words, this test bed configuration represents a modern option for the execution of one calculus when a large volume of Web text documents is necessary to build a category database.

The test bed environment for this stage of the study was designed while considering diverse situations. In the first scenario, the computational architecture was implemented with only one node working inside one core. The second configuration was characterized by two virtual machines (VMs), executing the Hadoop software environment in two different machines, with a unique core. The latter scenario represents an interesting distributed environment with one master and one slave node. Figure 4 below presents the main interface from the master VM executing Hadoop.

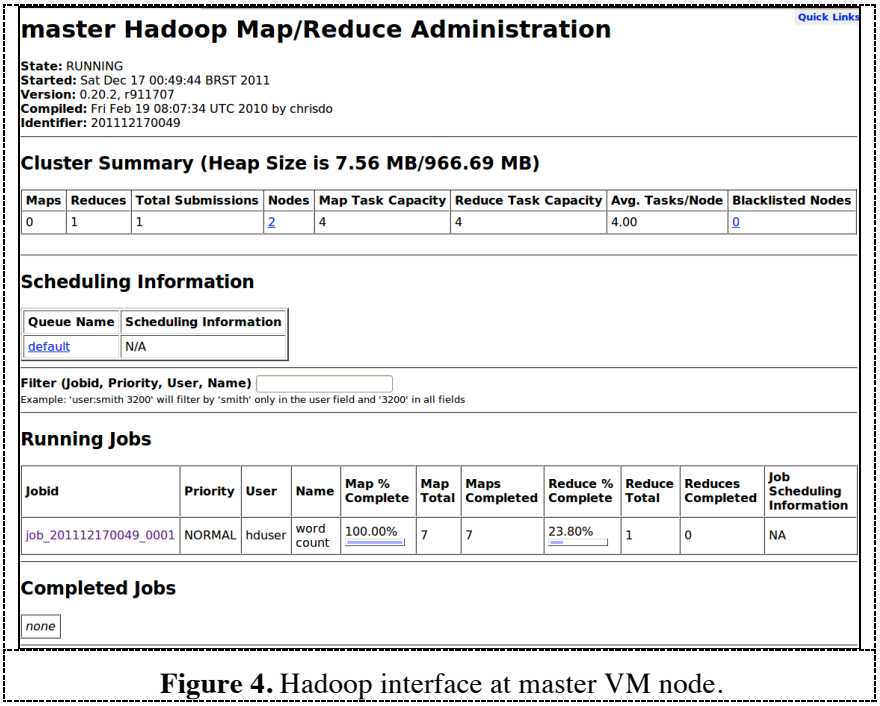

Figure 4. Hadoop interface at master VM node.

The Hadoop installation requires the Java Runtime Environment (JRE) and the Secure Shell (SSH) is also necessary. This package is used for initialization and finalization of script patterns created between nodes from the cluster. Compatible file systems with Hadoop inside the VMs inform slave node locations to the master node. Therefore, it will be possible to perform all main operations and task management of the environment.

As an ordinary procedure, the system performs executions of all local feasible operations. In other words, the priority is to execute any local task, where data exists locally. In the case of local data absence, the system will search for other configurations for the execution. The HDFS file system adopts a different data distributed storage policy to enhance environment performance.

One typical small Hadoop configuration, similar to the conceived test bed, comprises only one master node and several other slave nodes. In contrast, in a large Hadoop environment, the HDFS is managed through a unique and dedicated node that stores the NameNode and all file system indexes. In this environment, a secondary NameNode exists, which can mirror the structure execution of the master node, in case of failure from the master.

4. Empirical Results

In this section, results from our experiments utilizing the RapidMiner software package and the Hadoop/HDFS environment are presented. The objective is to have better assessment of the empirical process of text pre-processing phases to build term basis from a diffuse automatic document classification (ADC).

In the first configuration of our tests, the utilization of RapidMiner was considered, characterized by serial operations in an environment with one core. A set of four experiments were performed in this configuration, different among each due to text file sizes. In the first experiment, the operation was to process three different text files. These documents had sizes of 0.658MBytes, 1.356MBytes, and 1.56 MBytes, respectively. In the following test, the same text documents were processed. In the third execution, file sizes have been changed to 1.356 MBytes, 1.56 MBytes, and 13.6 MBytes, respectively. In the final experiment from this first set with the RapidMiner software package, the file sizes were changed once more to 1.56 Mbytes, 10.7 Mbytes, and 13.6 Mbytes. Figure 5 below shows

results from these four initial experiments. Different text documents, with different sizes, were submitted for execution.

In the research presented by Alves et al. [29] several aspects which could influence the success of a Web application are presented. However, response latency is by far the most important element to any user of any software tool. This point can be observed in Figure 5, taking into consideration the issues of size and elapsed time for the respective responses. In other words, all tasks were executed successfully in an acceptable time threshold by the RapidMiner software.

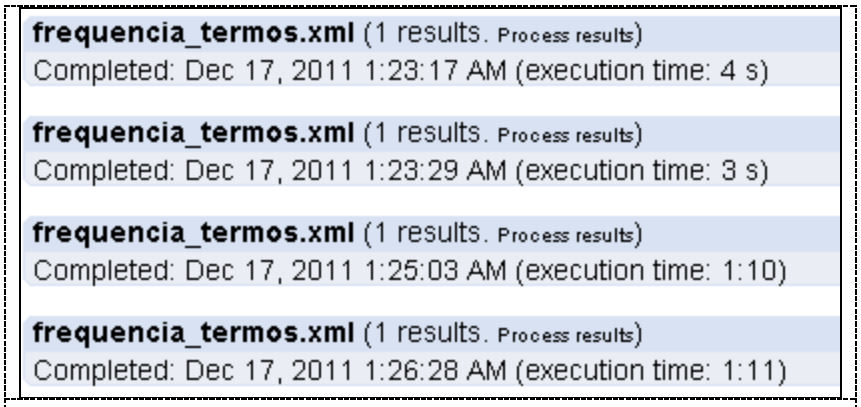

Figure 5. RapidMiner execution of four tests.

The second batch of experiments was to execute some case studies with the Hadoop environment. The first test considered, similarly to the RapidMiner experiment, one node acting as a master/slave, without data replication. In this test, documents had sizes of 0.658MBytes, 1.356MBytes, and 1.56 MBytes, respectively. The elapsed time to execute this task was 1 minute and 46 seconds, as illustrated in Figure 6 below.

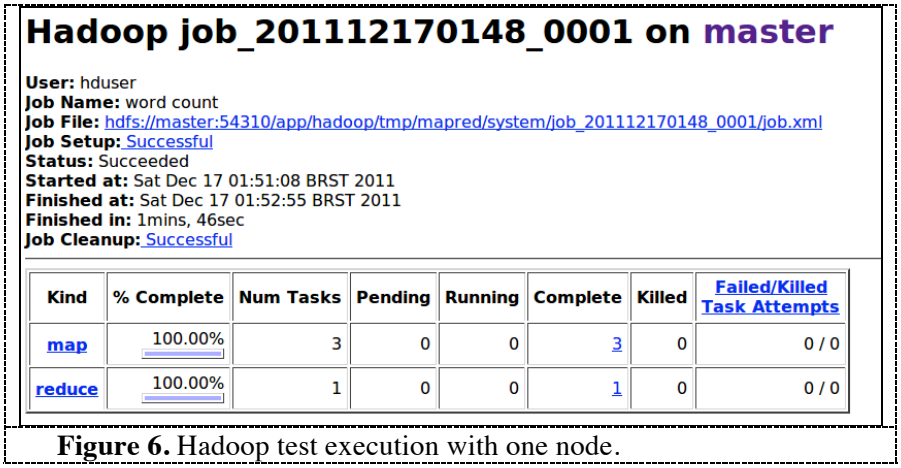

Figure 6. Hadoop test execution with one node.

In the second set of tests two different nodes were configured, where the same text documents (sized 0.658MBytes, 1.356MBytes, and 1.56 Mbytes) were processed. The resulting elapsed time was 32 seconds, as shown in Figure 7 below. The application failed to process other experiments similar to those performed by RapidMiner, with different files and different file sizes. This problem seems to be a hardware limitation where some nodes do not have enough memory and therefore have crashed.

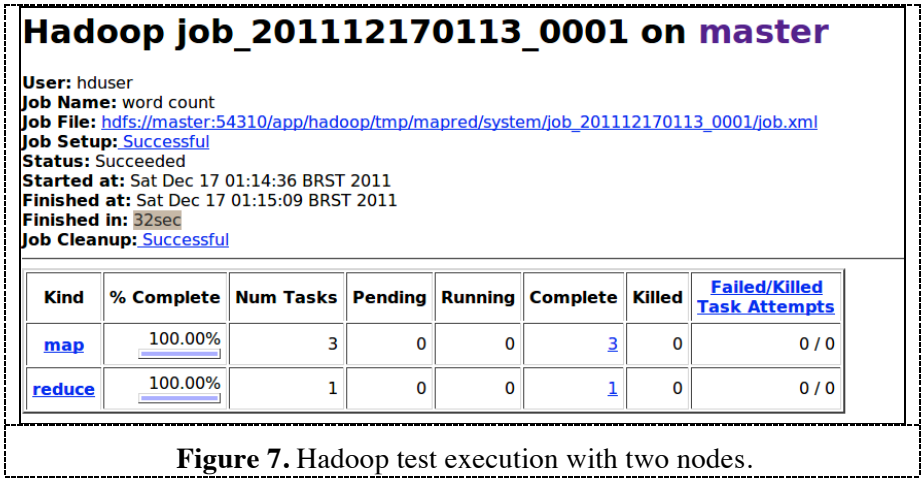

Figure 7. Hadoop test execution with two nodes.

5. Conclusion and Suggestions for Further Research

The goal of this case-comparison study is to exploit differences between serial and parallel paradigms, especially when a large volumes of data from Web documents is utilized to build a category database. In the literature, several studies point out that the distributed document processing of unstructured data have presented interesting levels of efficiency.

The tested database stores normalized terms within their category in order to assess their level of importance. This alternative proposal of Hadoop implementation was built for an Automatic Text Document Categorization System (ATDCS) based on two concepts: similarity and accuracy. These two concepts are input to a fuzzy modeling which ultimately allows specifying the level of relevance of a text to a specific category.

Preliminary results in the test environment did not present favorable performance increment for the distributed implementation if the volume of unstructured data is on the order of 10^6 (Mega Bytes). Still, it is estimated that Hadoop performs better when the amount of data is on the order of 10^15 (Peta Bytes). Normally there is the expectation that in parallel programs processing should be more efficient, since the process is split into parts, where each can be executed simultaneously. In cases where the volume of data is not significant, the MapReduce solution implemented by Hadoop is not more interesting than the serialized solution from RapidMiner.

References

[1] Royal Pingdom. [online] http://royal.pingdom.com/2012/01/17/internet-2011-in-numbers/ (Accessed 15 March 2013).

[2] Autonomy: an HP company. What is big data? [online] http://www.autonomy.com/content/Technology/what-is-big-data/index.en.html (Accessed 15 March 2013).

[3] F.J. Knaesel, M.A.R. Dantas. High-performance sharing of consistent data in ad hoc networks. *Int. J. of Networking and Virtual Organisations*, 2009 Vol.6, No.3, pp.259 – 269.

[4] Diogo R. Viegas, M.A.R. Dantas, Michael A. Bauer. A case study of transport protocols to improve the execution of applications in virtual organisations utilising multicluster network configurations. *Int. J. of Networking and Virtual Organisations*, 2010 Vol.7, No.5, pp.433 – 451.

[5] S. Sendhilkumar, G.S. Mahalakshmi, Context-based citation retrieval, *Int. J. of Networking and Virtual Organisations*. 2011 Vol.8, No.1/2, pp.98 – 122.

[6] Yang, Y; Pederson, J, 1997: A Comparative Study on Feature Selection in Text Categorization. *In Proceedings of 14th International Conference on Machine Learning*, Morgan Kaufmann Publishers, San Francisco, US (1997), 412-420.

[7] Russell, Stuart Jonathan, Norvig, Peter. 2009. Artificial intelligence: a modern approach. 3rd edition. Prentice-Hall
[8] Baeza-Yates, Ricardo, Ribeiro-Neto, Berthier. (1999) Modern Information Retrieval. 1 ed., United Kingdom, Addison-Wesley.
[9] RapidMiner. [online] http://rapid-i.com (Accessed 15 March 2013).
[10] Apache Hadoop. [online] http://hadoop.apache.org/ (Accessed 15 March 2013).
[11] Weka. [online] http://www.cs.waikato.ac.nz/ml/weka/index.html (Accessed 15 March 2013).
[12] Lammal, Ralf. Googles MapReduce Programming Model Revisited (2007). [online] http://logic.cse.unt.edu/tarau/teaching/SoftDev/docs/mapReduce.pdf (Accessed 15 March 2013).
[13] Google Code University. Introduction to Parallel Programming and MapReduce. [online] http://code.google.com/intl/pt-BR/edu/parallel/mapreduce-tutorial.html (Accessed 15 March 2013).
[14] Google File System - GFS. [online] http://research.google.com/archive/gfs.html (Accessed 15 March 2013).
[15] Facebook. [online] http://engineering.twitter.com/2010/04/hadoop-at-twitter.html (Accessed 15 March 2013).
[16] Last.fm. [online] http://www.lastfm.com.br/ (Accessed 15 March 2013).
[17] Twitter. https://twitter.com/ (Accessed 15 March 2013).
[18] International Business Machines (IBM). [online] http://www-01.ibm.com/software/data/infosphere/hadoop/ (Accessed 15 March 2013).
[19] Microsoft. [online] http://www.microsoft.com/en-us/sqlserver/solutions-technologies/business-intelligence/big-data.aspx (Accessed 15 March 2013).
[20] S3. [online] http://aws.amazon.com/pt/s3/ (Accessed 15 March 2013).
[21] Ceph. [online] http://ceph.com/ceph-storage/file-system (Accessed 15 March 2013).
[22] FhGFS. [online] http://www.fhgfs.com/cms/ (Accessed 15 March 2013).
[23] Lustre. [online] http://wiki.lustre.org/index.php/Main_Page (Accessed 15 March 2013).
[24] GPFS. [online] http://www-03.ibm.com/systems/software/gpfs/ (Accessed 15 March 2013).
[25] PVFS. [online] http://www.pvfs.org/ (Accessed 15 March 2013).
[26] HDFS Architecture Guide. Apache Software Foundation. [online] http://hadoop.apache.org/common/docs/current/hdfs_design.html (Accessed 15 March 2013).
[27] MapReduce. [online] http://research.google.com/archive/mapreduce.html (Accessed 15 March 2013).
[28] Goller, G., J. Loning, T. Will, W. Wolff. Automatic Document Classification: A thorough Evaluation of Various Methods. *Internationalen Symposiums für Informationswissenschaft*, Nov. 2000, pp. 145-162.
[29] Alves, V., Dantas, M.A.R., An Approach Based on Speedup Web Paradigm, submitted to Webmedia 2013.